Signal Processing in the Pure Programming Language

Albert Gräf

Dept. of Music Informatics

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Full paper, slides, examples at:
http://pure-lang.googlecode.com/svn/docs/

- Pure, the programming language
- Signal processing with Pure
- The future of Pure

1. Pure, the Language

```
motion (x,y) (vx,vy) (ax,ay) (step dt:next)
= [x,y,vx,vy] :
  motion (x1,y1) (vx1,vy1) (ax,ay) next &
when
  vx = if abs x > 3 then -vx else vx;
  vy = if y < -3 then -vy else vy;
  x1 = x+dt*vx+dt*dt*ax/2; y1 = y+dt*vy+dt*dt*ay/2;
  vx1 = vx+dt*ax; vy1 = vy+dt*ay;
end;

motion _ (vx,vy) (ax,ay) (mouse x y:next)
= () : motion (x,y) (vx,0) (ax,ay) next &;

using actor;
ball = actor (motion (-3,3) (0.5,0) (0,-3));
```

Lessons learned from Q

+ *Term rewriting* makes for a nice algebraic/functional programming language.
+ *Dynamic typing* makes it convenient to interface to interpreted realtime environments (Pd, SuperCollider, ...).
– Bytecode interpretation was *slow*. People kept bugging me for a compiler. (Wanted one myself, actually.)
– Q lacked some key features, most notably *local functions*.
– The global mutex. **Boo, hiss!**

⇒ A new implementation was needed.

LLVM changed the game

- **LLVM** = "Low-Level Virtual Machine" (llvm.org).
- Generate *native, optimized code* in a platform-independent way (LLVM IR).
- Built-in **JIT** ("Just In Time") compilation.
- *Batch-compile* programs to fast native code.
- Dead easy to *interface to C*.
- "Compiled scripting language."
- Gave the language a facelift along the way.

Before:

```
gr P X Y          = P Y X;
lq P X Y          = not gr P X Y;

qsort P []        = [];
qsort P [X|Xs]    = qsort P (filter (gr P X) Xs) ++
                    [X|qsort P (filter (lq P X) Xs)];
```

Main influences: Pascal, Prolog, Miranda

After:

```
qsort p []        = [];
qsort p (x:xs)    = qsort p [l | l = xs; l<x] +
                    (x : qsort p [r | r = xs; r>=x])
                    with x<y = p x y; x>=y = ~p x y end;
```

Main influences: Haskell, Aardappel, Alice ML

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Features

- *Algebraic/functional* programming language.
- *Term rewriting* + *modern FP* (lambda, currying, closures).
- *Lists* and Octave-style *matrices*, list and matrix *comprehensions*.
- *Dynamic typing*, terms as data.
- *Eager* + *lazy* evaluation.
- Easy *C interface*. (Not really *that* pure!)
- In the planning stage: *concurrency*.

```
fact 0 = 1;
fact n = n*fact (n-1) if n>0;
```

```
x:y:xs = y:x:xs if x>y;
       = x:xs   if x==y;
```

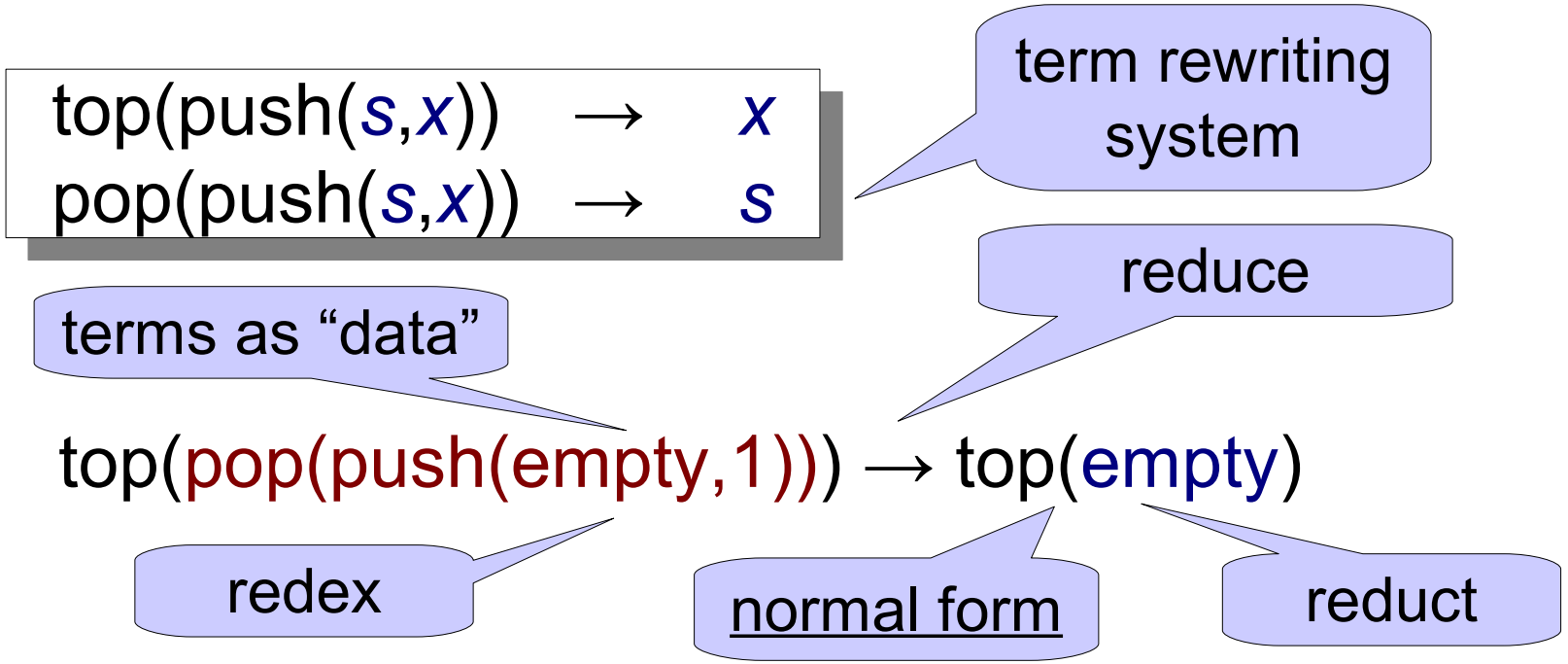
```
tri n m = [x,y   | x = 1..n; y = 1..m; x<y];
eye n   = {i==j | i = 1..n; j = 1..n};
```

```
insert nil y          = bin y nil nil;
insert (bin x l r) y = bin x (insert l y) r if y<x;
                      = bin x l (insert r y);
```

```
primes = sieve (2..inf) with
  sieve (p:qs) = p : sieve [q | q = qs; q mod p] &;
end;
```

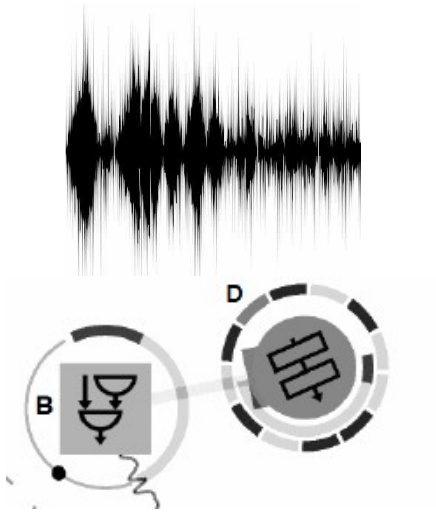
```
extern int rand();
[rand | i = 1..20];
```

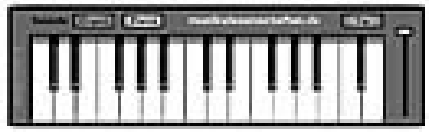
What's this term rewriting?



- Whitehead et al: *universal algebra, equational logic*
- O'Donnell et al: term rewriting as *programming language*
- Goguen, Mahr et al: *algebraic specification*
- Milner, Turner et al: *modern functional programming*

2. Signal Processing





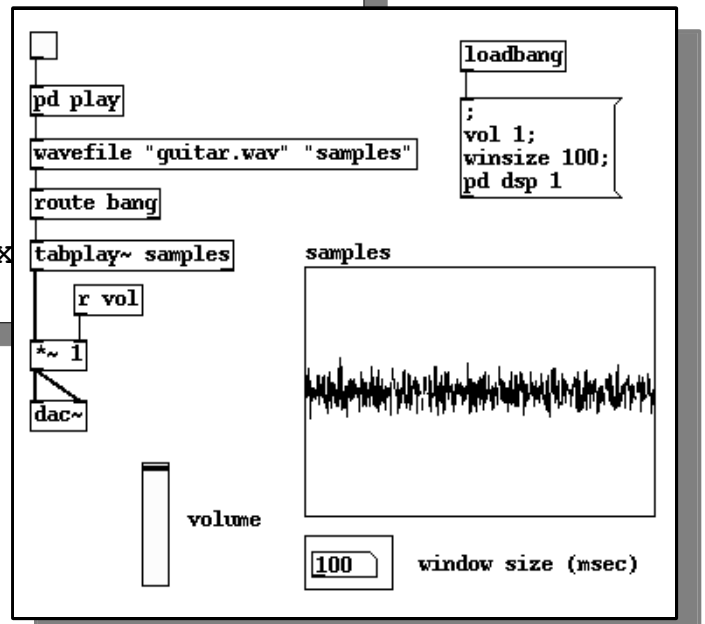
Actor-style processing

```
wavefile fname aname = process with
  process reset = () when sf_seek fp 0 0; end;
  process bang  = if ok res then bang else () when
    n = nsamples; wave = dmatrix n;
    res = sf_read_double fp wave n;
    pd_setbuffer aname wave;
  end;
  nsamples      = pd_getbuffersize aname;
  ok res        = bigintp res && res>0;
end when
  fp::pointer = sentry sf_close
                (sf_open fname 0x10 (imatrix
end;
```

create buffer

read wave

copy into Pd array



`wavefile.pure`: Use libsndfile to shovel chunks from a wave file into a Pd audio buffer.

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

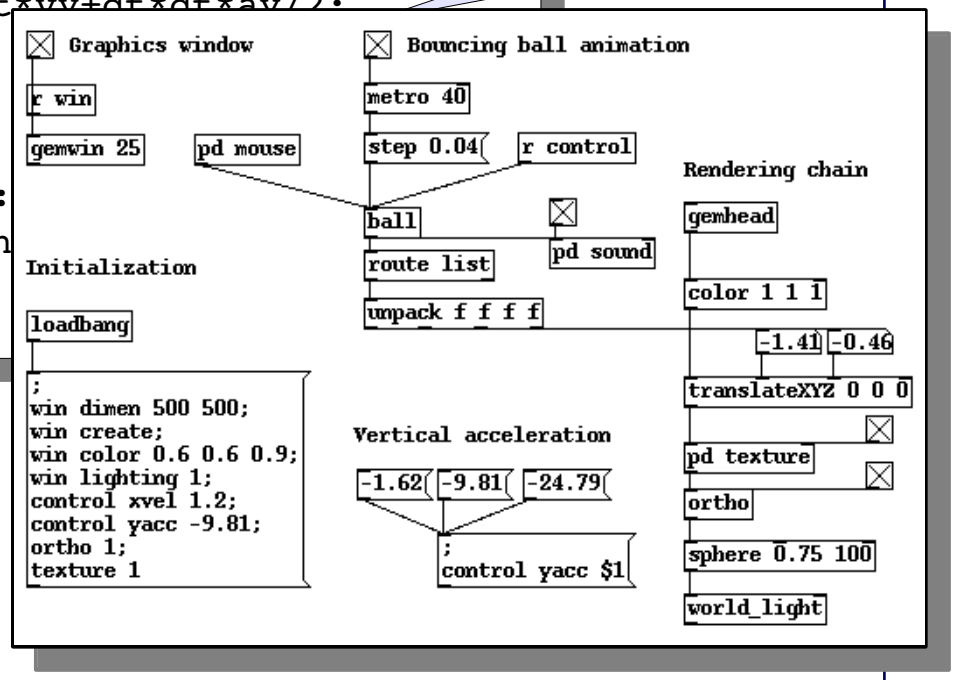
Stream processing

input stream gets mapped to output stream

collision detection

Newtonian motion

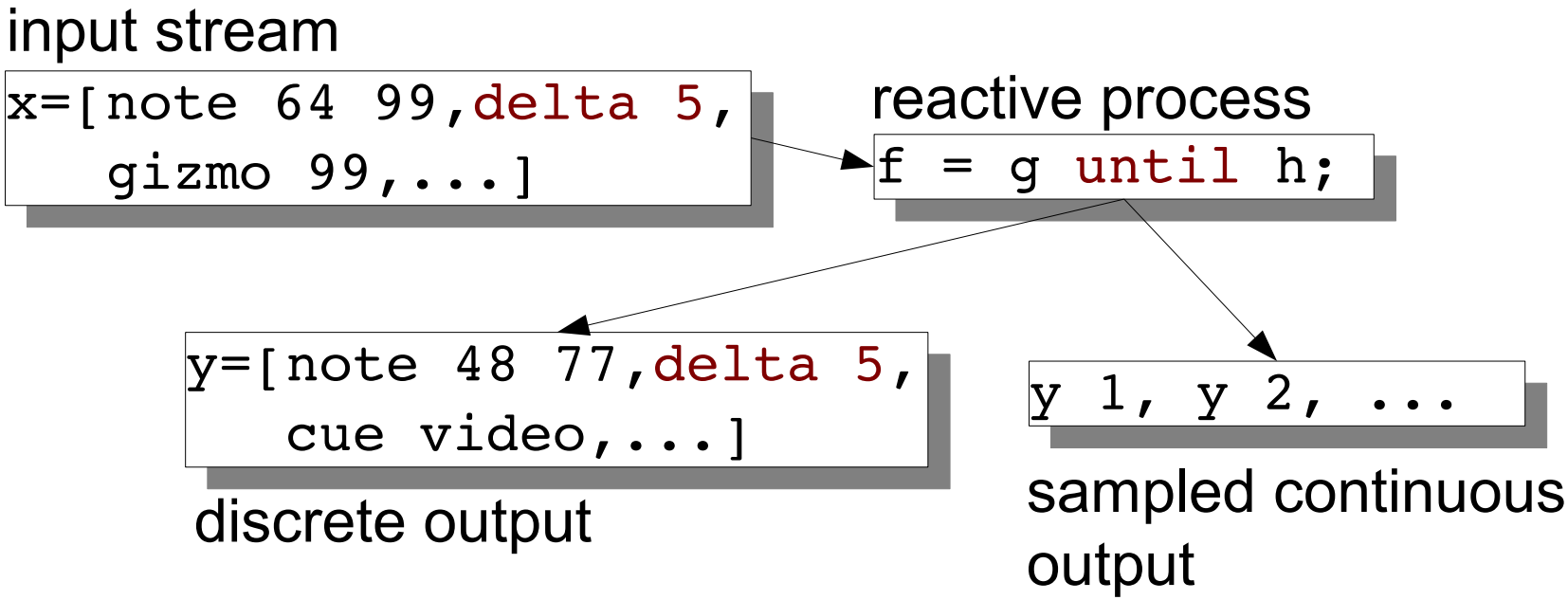
```
motion (x,y) (vx,vy) (ax,ay) (step dt:next)
= [x,y,vx,vy] : motion (x1,y1) (vx1,vy1) (ax,ay) next &
when
  vx = if abs x > 3 then -vx else vx;
  vy = if y < -3 then -vy else vy;
  x1 = x+dt*vx+dt*dt*ax/2; y1 = y+dt*vy+dt*dt*ay/2;
  vx1 = vx+dt*ax; vy1 = vy+dt*ay;
end;

motion _ (vx,vy) (ax,ay) (mouse x y:
= () : motion (x,y) (vx,0) (ax,ay) n

ball = actor (motion (-3,3) (0.5,0)
```



`ball.pure`: Bouncing ball animation using Gem.

3. The Future...

Functional Reactive Programming



- Inventend (mostly) at Yale (Hudak et al). FPL: Haskell.
- Elegant **algebraic semantics**. But: No “standard” algebra for asynchronous processes yet! Much more complex than the synchronous case.

Pure as a Testbed for Signal Algebras

- Play with different algebraic models.
  - Built-in support for streams and HOFs.
  - Make our own algebras (constants, functions, operators).
  - Symbolic rewriting rules to specify semantics.
  - Dynamic typing lets us handle ad-hoc event structures.
- Interface with “the world out there”.
  - databases, XML, realtime engines, GUI, graphics, ...

```
infixl 1 until;
(x until y) []     = [];
(x until y) (a:z) = y a z;
```

Other stuff to be done

- Compiler improvements (better code for numerics, more aggressive optimizations).
- Concurrency: Data parallelism (parallel matrix comprehensions), concurrent futures.
- Pd-Pure: Add audio objects.
- Interfaces to Faust, Max, SuperCollider.
- Whatever comes up on the Pure mailing list... http://groups.google.com/group/pure-lang